

09/21/99

jc674 U.S. PTO

# UTILITY PATENT APPLICATION TRANSMITTAL (Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.  
ST9-99-093

Total Pages in this Submission

76

**TO THE ASSISTANT COMMISSIONER FOR PATENTS**

Box Patent Application

Washington, D.C. 20231

Transmitted herewith for filing under 35 U.S.C. 111(a) and 37 C.F.R. 1.53(b) is a new utility patent application for an invention entitled:

**THE ARCHITECTURE AND IMPLEMENTATION OF A DYNAMIC RMI SERVER CONFIGURATION  
HIERARCHY TO SUPPORT FEDERATED SEARCH AND UPDATE ACROSS HETEROGENEOUS  
DATASTORES**

and invented by:

Kehsing J. Chou, et al.

jc678 U.S. PTO

09/30/96

09/21/99

If a **CONTINUATION APPLICATION**, check appropriate box and supply the requisite information:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: \_\_\_\_\_

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: \_\_\_\_\_

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: \_\_\_\_\_

Enclosed are:

**Application Elements**

1. ☒ Filing fee as calculated and transmitted as described below
2. ☒ Specification having 65 pages and including the following:
  - a. ☒ Descriptive Title of the Invention
  - b. ☒ Cross References to Related Applications (if applicable)
  - c. ☐ Statement Regarding Federally-sponsored Research/Development (if applicable)
  - d. ☐ Reference to Microfiche Appendix (if applicable)
  - e. ☒ Background of the Invention
  - f. ☒ Brief Summary of the Invention
  - g. ☒ Brief Description of the Drawings (if drawings filed)
  - h. ☒ Detailed Description
  - i. ☒ Claim(s) as Classified Below
  - j. ☒ Abstract of the Disclosure

**UTILITY PATENT APPLICATION TRANSMITTAL**  
**(Large Entity)**

*(Only for new nonprovisional applications under 37 CFR 1.33(b))*

Docket No.  
ST9-99-093

Total Pages in this Submission  
**76**

**Application Elements (Continued)**

3. ☒ Drawing(s) *(when necessary as prescribed by 35 USC 113)*
- a. ☐ Formal                      Number of Sheets \_\_\_\_\_
- b. ☒ Informal                      Number of Sheets 9
4. ☒ Oath or Declaration
- a. ☐ Newly executed *(original or copy)*                      ☒ Unexecuted
- b. ☐ Copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional application only)*
- c. ☒ With Power of Attorney                      ☐ Without Power of Attorney
- d. ☐ DELETION OF INVENTOR(S)  
Signed statement attached deleting inventor(s) named in the prior application,  
see 37 C.F.R. 1.63(d)(2) and 1.33(b).
5. ☐ Incorporation By Reference *(usable if Box 4b is checked)*  
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under  
Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby  
incorporated by reference therein.
6. ☐ Computer Program in Microfiche *(Appendix)*
7. ☐ Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all must be included)*
- a. ☐ Paper Copy
- b. ☐ Computer Readable Copy *(identical to computer copy)*
- c. ☐ Statement Verifying Identical Paper and Computer Readable Copy

**Accompanying Application Parts**

8. ☐ Assignment Papers *(cover sheet & document(s))*
9. ☐ 37 CFR 3.73(B) Statement *(when there is an assignee)*
10. ☐ English Translation Document *(if applicable)*
11. ☐ Information Disclosure Statement/PTO-1449                      ☐ Copies of IDS Citations
12. ☐ Preliminary Amendment
13. ☒ Acknowledgment postcard
14. ☒ Certificate of Mailing
- ☐ First Class                      ☒ Express Mail *(Specify Label No.):* EL243341245US

**UTILITY PATENT APPLICATION TRANSMITTAL**  
**(Large Entity)**

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.  
ST9-99-093

Total Pages in this Submission

76

**Accompanying Application Parts (Continued)**

15. ☐ Certified Copy of Priority Document(s) (if foreign priority is claimed)
16. ☐ Additional Enclosures (please identify below):

**Fee Calculation and Transmittal**

**CLAIMS AS FILED**

For	#Filed	#Allowed	#Extra	Rate	Fee
Total Claims	18	- 20 =	0	x \$18.00	\$0.00
Indep. Claims	3	- 3 =	0	x \$78.00	\$0.00
Multiple Dependent Claims (check if applicable) <input type="checkbox"/>					\$0.00
BASIC FEE					\$760.00
OTHER FEE (specify purpose)					\$0.00
TOTAL FILING FEE					\$760.00

- ☐ A check in the amount of \_\_\_\_\_ to cover the filing fee is enclosed.
- ☒ The Commissioner is hereby authorized to charge and credit Deposit Account No. 09-0460 as described below. A duplicate copy of this sheet is enclosed.
- ☒ Charge the amount of \$760.00 as filing fee.
- ☒ Credit any overpayment.
- ☒ Charge any additional filing fees required under 37 C.F.R. 1.16 and 1.17.
- ☐ Charge the issue fee set in 37 C.F.R. 1.18 at the mailing of the Notice of Allowance, pursuant to 37 C.F.R. 1.311(b).

  
Signature

Dated: September 21, 1999

Janaki Komanduri, Reg. No. 40,684  
PRETTY, SCHROEDER & POPLAWSKI  
444 S. Flower Street, 19th Floor  
Los Angeles, California 90071  
Tel: 213-622-7700  
Fax: 213-489-4210

cc:

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Kehsing J. Chou, et al. Art Unit: Not Assigned  
Serial No.: Not Assigned Examiner: Not Assigned  
Filed: September 21, 1999  
For: THE ARCHITECTURE AND IMPLEMENTATION OF A DYNAMIC RMI  
SERVER CONFIGURATION HIERARCHY TO SUPPORT  
FEDERATED SEARCH AND UPDATE ACROSS HETEROGENEOUS  
DATASTORES

CERTIFICATE OF MAILING VIA U.S. EXPRESS MAIL

Date of Deposit: September 21, 1999

BOX PATENT APPLICATION  
Assistant Commissioner for Patents  
Washington, D.C. 20231

Dear Sir:

I hereby certify that

- ☒ Letter of transmittal in duplicate
- ☒ patent application (55 pages of specification; 18 claims
- ☒ 9 sheets of informal drawings
- ☒ unsigned Declaration and Power of Attorney
- ☒ return postcard

are being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service with sufficient postage under 37 CFR 1.10 on the date indicated above and are addressed to:

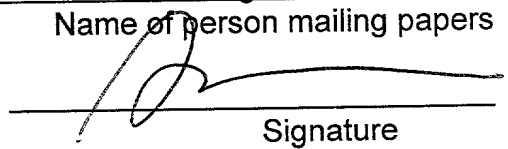
BOX PATENT APPLICATION  
Assistant Commissioner for Patents  
Washington, D.C. 20231.

September 21, 1999

Date of Deposit

Sherlin Yaghoubzadeh

Name of person mailing papers

  
Signature

UTILITY APPLICATION

OF

Kehsing J. Chou, Mei-Ing W. Huang, Taejae Lee,  
Basuki N. Soetarman, Robert N. Summers, and Mimi P. Vo

FOR

UNITED STATES LETTERS PATENT

on

THE ARCHITECTURE AND IMPLEMENTATION OF A DYNAMIC RMI SERVER  
CONFIGURATION HIERARCHY TO SUPPORT FEDERATED SEARCH  
AND UPDATE ACROSS HETEROGENEOUS DATASTORES

Docket No. ST9-99-093 / P70 43161

Drawings: 9

Attorneys

PRETTY, SCHROEDER & POPLAWSKI

444 South Flower Street, 19<sup>th</sup> Floor

Los Angeles, California 90071

Ofc: 213/622-7700

Fax: 213/489-4210

CERTIFICATE OF MAILING BY "EXPRESS MAIL"

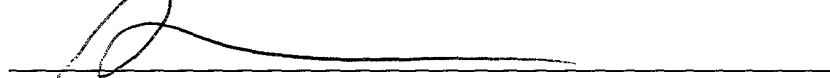
"EXPRESS MAIL" MAILING LABEL NUMBER EL243341245US

DATE OF DEPOSIT September 21, 1999

I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL  
SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR 1-10 ON THE DATE  
INDICATED ABOVE AND IS ADDRESSED TO BOX PATENT APPLICATION, THE ASSISTANT  
COMMISSIONER FOR PATENTS, WASHINGTON, D. C. 20231

Sherlin Yaghoubzadeh

(TYPED OR PRINTED NAME OF PERSON MAILING PAPER OR FEE)



(SIGNATURE OF PERSON MAILING PAPER OR FEE)

THE ARCHITECTURE AND IMPLEMENTATION OF A DYNAMIC RMI SERVER  
CONFIGURATION HIERARCHY TO SUPPORT FEDERATED SEARCH  
AND UPDATE ACROSS HETEROGENEOUS DATASTORES

CROSS-REFERENCE TO RELATED APPLICATIONS

5 This application is related to the following co-pending and commonly-assigned patent applications:

Application No. --/--,---, entitled "MANAGING RESULTS OF FEDERATED SEARCHES ACROSS HETEROGENEOUS DATASTORES WITH A FEDERATED RESULT SET CURSOR OBJECT," filed on same date herewith, by Basuki N. Soertarman et al., attorney's  
10 docket number ST9-99-079;

Application No. --/--,---, entitled "THE DESIGN AND IMPLEMENTATION OF A CLIENT/SERVER FRAMEWORK FOR FEDERATED MULTI-SEARCH AND UPDATE ACROSS HETEROGENEOUS DATASTORES," filed on same date herewith, by Kehsing J. Chou et al., attorney's docket number ST9-99-094;

15 Application No. --/--,---, entitled "ARCHITECTURE TO ENABLE SEARCH GATEWAYS AS PART OF FEDERATED SEARCH," filed on same date herewith, by Kehsing J. Chou et al., attorney's docket number ST9-99-097;

Application No. --/--,---, entitled "DELAYED DELIVERY OF QUERY RESULTS OR OTHER DATA FROM A FEDERATED SERVER TO A FEDERATED CLIENT UNTIL SUCH  
20 INFORMATION IS NEEDED," filed on same date herewith, by M. W. Huang et al., attorney's docket number ST9-99-123;

Application No. --/--,---, entitled "IMPROVED EXTENDED DATA OBJECT ARCHITECTURE FOR HANDLING MULTI-STREAMING AND COMPLEX MULTI-DIMENSIONAL FILES," filed on same date herewith, by Kehsing J. Chou et al., attorney's  
25 docket number ST9-99-139;

Application No. 08/852,062, entitled "CLASS HIERARCHY FOR OBJECT AGGREGATION REPRESENTATION OF RELATIONAL DATABASE ROWS WITH CELLS HAVING NONTRADITIONAL DATATYPES," filed on May 6, 1997, by Daniel T. Chang et al., attorney's docket number ST9-97-032;

Application No. 08/852,055, entitled "FLEXIBLE OBJECT REPRESENTATION OF RELATIONAL DATABASE CELLS HAVING NONTRADITIONAL DATATYPES," filed on May 6, 1997, by Daniel T. Chang et al., attorney's docket number ST9-97-033;

Application No. 09/052,678, entitled "MANAGING RESULTS OF FEDERATED SEARCHES ACROSS HETEROGENEOUS DATASTORES WITH A FEDERATED COLLECTION OBJECT," filed on April 1, 1998, by Daniel T. Chang et al., attorney's docket number ST9-98-016;

Application No. 09/052,680, entitled "FEDERATED SEARCHING OF HETEROGENEOUS DATASTORES USING A FEDERATED DATASTORE OBJECT," filed on April 1, 1998, by Daniel T. Chang et al., attorney's docket number ST9-98-017; and

Application No. 09/052,679, entitled "FEDERATED SEARCHING OF HETEROGENEOUS DATASTORES USING A FEDERATED QUERY OBJECT," filed on April 1, 1998, by Daniel T. Chang et al., attorney's docket number ST9-98-018;

each of which is incorporated by reference herein.

## BACKGROUND OF THE INVENTION

### 1. Field of the Invention.

This invention relates in general to database management systems performed by computers, and in particular, to an architecture and implementation of a dynamic Remote Method Invocation (RMI) server configuration hierarchy to support federated search and update across heterogeneous datastores.

### 2. Description of Related Art.

The present invention relates to a system and method for representing and searching multiple heterogeneous datastores and managing the results of such searches. Datastore is a term used to refer to a generic data storage facility, such as a relational data base, flat-file, hierarchical data base, etc. Heterogeneous is a term used to indicate that the datastores need not be similar to each other. For example, each datastore may store different types of data, such as image or text, or each datastore may be based on a different theory of data model, such as Digital Library/VisualInfo or Domino Extended Search (DES).

For nearly half a century computers have been used by businesses to manage information such as numbers and text, mainly in the form of coded data. However, business data represents only a small part of the world's information. As storage, communication and information processing technologies advance, and as their costs come down, it becomes more feasible to digitize other various types of data, store large volumes of it, and be able to distribute it on demand to users at their place of business or home.

New digitization technologies have emerged in the last decade to digitize images, audio, and video, giving birth to a new type of digital multimedia information. These multimedia objects are quite different from the business data that computers managed in the past, and often require more advanced information management system infrastructures with new capabilities. Such systems are often called "digital libraries."

Bringing new digital technologies can do much more than just replace physical objects with their electronic representation. It enables instant access to information; supports fast, accurate, and powerful search mechanisms; provides, new "experiential" (i.e. virtual reality) user interfaces; and implements new ways of protecting the rights of information owners. These properties make digital library solutions even more attractive and acceptable not only to corporate IS organizations, but to the information owners, publishers and service providers.

Generally, business data is created by a business process (an airline ticket reservation, a deposit at the bank, and a claim processing at an insurance company are examples). Most of these processes have been automated by computers and produce business data in digital form (text and numbers). Therefore it is usually structured coded data. Multimedia data, on the contrary, cannot be fully pre-structured (its use is not fully predictable) because it is the result of the creation of a human being or the digitization of an object of the real world (x-rays, geophysical mapping, etc.) rather than a computer algorithm.

The average size of business data in digital form is relatively small. A banking record -- including a customers name, address, phone number, account number, balance, etc.--represents at most a few hundred characters, i.e. few hundreds/thousands of bits. The digitization of multimedia information (image, audio, video) produces a large set of bits called an "object" or "blobs" (Binary Large Objects). For example, a digitized image of the parchments from the Vatican Library takes as much as the equivalent of 30 million characters (30 MB) to be stored.



The digitization of a movie, even after compression, may take as much as the equivalent of several billions of characters (3-4 GB) to be stored.

Multimedia information is typically stored as much larger objects, ever increasing in quantity and therefore requiring special storage mechanisms. Classical business computer systems have not been designed to directly store such large objects. Specialized storage technologies may be required for certain types of information, e.g. media streamers for video or music. Because certain multimedia information needs to be preserved "forever" it also requires special storage management functions providing automated back-up and migration to new storage technologies as they become available and as old technologies become obsolete.

Finally, for performance reasons, the multimedia data is often placed in the proximity of the users with the system supporting multiple distributed object servers. This often requires a logical separation between applications, indices, and data to ensure independence from any changes in the location of the data.

The indexing of business data is often imbedded into the data itself. When the automated business process stores a person's name in the column "NAME," it actually indexes that information. Multimedia information objects usually do not contain indexing information. This "meta data" needs to be created in addition by developers or librarians. The indexing information for multimedia information is often kept in "business like" databases separated from the physical object.

In a Digital Library (DL), the multimedia object can be linked with the associated indexing information, since both are available in digital form. Integration of this legacy catalog information with the digitized object is crucial and is one of the great advantages of DL technology. Different types of objects can be categorized differently as appropriate for each object type. Existing standards like MARC records for libraries, Finding Aids for archiving of special collections, etc... can be used when appropriate.

The indexing information used for catalog searches in physical libraries is mostly what one can read on the covers of the books: authors name, title, publisher, ISBN,... enriched by other information created by librarians based on the content of the books (abstracts, subjects, keywords,...). In digital libraries, the entire content of books, images, music, films, etc.. are available and "new content" technologies are needed; technologies for full text searching, image

content searching (searching based on color, texture, shape, etc...), video content searching, and audio content searching. The integrated combination of catalog searches (e.g. SQL) with content searches will provide more powerful search and access functions. These technologies can also be used to partially automate further indexing, classification, and abstracting of objects based on content.

To harness the massive amounts of information spread throughout these networks, it has become necessary for a user to search numerous storage facilities at the same time without having to consider the particular implementation of each storage facility.

Object-oriented approaches are generally better suited for such complex data management. The term "object-oriented" refers to a software design method which uses "classes" and "objects" to model abstract or real objects. An "object" is the main building block of object-oriented programming, and is a programming unit which has both data and functionality (i.e., "methods"). A "class" defines the implementation of a particular kind of object, the variables and methods it uses, and the parent class it belongs to.

Some known programming tools that can be used for developing search and result-management frameworks include IBM VisualAge C++, Microsoft Visual C++, Microsoft Visual J++, and Java.

There is a need in the art for an improved federated system. In particular, there is a need in the art for an architecture and implementation of a dynamic Remote Method Invocation (RMI) server configuration hierarchy to support federated search and update across heterogeneous datastores.

## SUMMARY OF THE INVENTION

To overcome the limitations in the prior art described above, and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention discloses a method, apparatus, and article of manufacture for an architecture and implementation of a dynamic Remote Method Invocation (RMI) server configuration hierarchy to support federated search and update across heterogeneous datastores.

According to an embodiment of the invention, the RMI server configuration hierarchy supports searching for data in one or more heterogeneous data sources within a computer system.

A request for data is received at a federated data source. Then, a server is selected to process the request based on a load of the server and based on whether the server can satisfy the request for data.

## BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 is a diagram illustrating a computer architecture that could be used in accordance with the present invention;

FIG. 2 is a diagram illustrating a class hierarchy for Data Object classes;

FIG. 3 is a diagram illustrating a class hierarchy for Datastore classes;

FIG. 4 is a diagram illustrating one composition of a federated datastore;

FIG. 5 is a diagram of an extended Grand Portal architecture;

FIG. 6 is a diagram illustrating individual datastores and federated compositions;

FIG. 7 is a diagram illustrating Remote Method invocation (RMI) client/server hierarchy;

FIG. 8 is a flow diagram of one use of the RMI architecture; and

FIG. 9 is a flow diagram illustrating searching within a RMI server hierarchy.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following description of the preferred embodiment, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural and functional changes may be made without departing from the scope of the present invention.

### Federated Architecture

FIG. 1 is a diagram illustrating a computer architecture that could be used in accordance with the present invention. The present invention is described herein by way of example and is not intended to be limited to the described embodiment. The description of the preferred

embodiment is based on, but certainly not limited to, the IBM design of Java Grand Portal Class Library, the Digital Library Java Application Programming Interface (API).

The Java Grand Portal 120 is comprised of client and server classes. In particular, Java Grand Portal is a set of Java classes which provides access and manipulation of local or remote data stored in Digital Library storage facilities. It uses a Java APIs based on OMG-Object Query Services (OQS) and a Dynamic Data Object protocol, which is a part of OMG/Persistence Object Services.

The Java APIs provide multi-search capabilities such as:

1. Searching within a given datastore using one or a combination of supported query types, i.e.

Parametric query - Queries requiring an exact match on the condition specified in the query predicate and the data values stored in the datastore.

Text query - Queries on the content of text fields for approximate match with the given text search expression, e.g. the existence (or non-existence) of certain phrases or word-stems.

Image query - Queries on the content of image fields for approximate match with the given image search expression, e.g. image with certain degree of similarity based on color percentages, layout, or texture.

2. Each search type is supported by one or more search-engines.
3. Searching on the results of a previous search.
4. Searching involving heterogeneous datastores.

661250"96966660

The Digital Library Grand Portal classes provide a convenient API for Java application users; the applications can be located at local or remote sites. Java classes will typically reside on both server and client sides; both sides providing the same interface. The client side of Java classes communicates with the server side to access data in the Digital Library through the network.

5 Communication between client and server sides is done by these classes; it is not necessary to add any additional programs.

In particular, FIG 1 is an architectural diagram outlining the structure of the federated search for Digital Library repositories using the federated datastore 100, comprised of a federated datastore client and server. A federated datastore 100 is a virtual datastore which combines

10 several heterogeneous datastores 102 into a consistent and unified conceptual view. This view, or a federated schema, is established via schema mapping 104 of the underlying datastores. The users interact with a federated datastore 100 using the federated schema, without needing to know about the individual datastores 102 which participate in the federated datastore 100.

One embodiment of the invention provides an architecture and implementation of a

15 dynamic Remote Method Invocation (RMI) server configuration hierarchy to support federated search and update across heterogeneous datastores. In one embodiment of the invention, one or more classes implement the architecture, and one or more methods are provided to manipulate the dynamic RMI server configuration hierarchy. In one embodiment, the class definitions and methods reside at one or more federated datastores and at one or more RMI servers.

20 The federated datastore 100 does not have a corresponding back-end client. Since it is a virtual datastore, the federated datastore 100 relies on the underlying physical back-end client associated with it, such as the DL client (i.e., Digital Library client), OnDemand, VisualInfo, DB2 etc. Digital Library, OnDemand, VisualInfo, and DB2 are all products from International Business Machines Corporation. As mentioned before, this association is established by a schema mapping

25 component 104.

The communication between the federated datastore 100 client and server can be done by any appropriate protocol. On top of Java Grand Portal client classes, the users can develop application programs using, for example, any existing Java Beans 122 development environment.

The federated datastore 100 coordinates query evaluation, data-access, and transaction

30 processing of the participating heterogeneous datastores 102. Given the federated schema, a multi-

search query can be formulated, executed, and coordinated to produce results in the form of a datastore-neutral dynamic data object.

Note that each heterogeneous datastore and the federated datastore are created using one datastore definition or superclass. The federated datastore 100 and the heterogeneous datastores 102 are all subclasses of a class called Datastore, therefore, all of these datastores 100 and 102 have the same interface. Therefore, a user would be able to access the federated datastore 100 and the heterogeneous datastores 102 in a consistent and uniform manner.

Additionally, the objects stored in the federated datastore 100 and the heterogeneous datastores 102 are subclasses of a Data Object class. The Data Object class includes subclasses for dynamic data objects (DDOs) and extended data objects (XDOs). A DDO has attributes, with type, value, and properties. The value of an attribute can be a reference to another DDO or XDO, or a collection of DDOs or XDOs.

FIG. 2 is a diagram illustrating a class hierarchy for Data Object classes. The objects stored in and manipulated by the datastores and fetch operations belong to data object classes. These objects are returned as the result of a fetch, or created and used in CRUD (add, retrieve, update, delete) operations.

A DataObjectBase 200 is an abstract base class for all data objects known by datastores. It has a protocol attribute, that indicates to the datastore which interface can be used to operate on this object. A XDOBase 210 is the base class used to represent user-defined-types (UDT) or large objects. In particular, the XDOBase 210 is the base class for some user-defined types 212 and XDOs 214. A XDO 214 represents complex UDTs or large objects (LOB). This object can exist stand-alone or as a part of a DDO 236. Therefore, it has a persistent object identifier and CRUD operations capabilities.

Blob 216 is a base class for BLOBs as a placeholder to share all generic operations pertaining to BLOBs. Clob 218 is a base class for CLOBs (Character Large Objects) as a placeholder to share all generic operations pertaining to CLOBs. DBClob 220 is a base class for DBCLOBs (database character large object) as a placeholder to share all generic operations pertaining to DBCLOBs. BlobDB2 222 represents a BLOB specific to DB2, and BlobDL 22 represents a BLOB specific to DL. Similarly, though not shown, there may be subclasses for ClobDB2, ClobDL, etc.

A DataObject 230 is a base class for PersistentObject 232 and DDOBase 234. A PersistentObject 232 represents a specific object whose code is statically generated and compiled. This type of object will not be covered in this document. A DDOBase 234 is a base class for a dynamic data object 236 (without the CRUD methods). A DDO (Dynamic Data Object) 236 represents generic data objects which are constructed dynamically at runtime. This object fits well with query and browsing activities in Grand Portal where objects are only known and generated at runtime. It supports the CRUD operations (add, retrieve, update, and delete), and, with the help of its associated datastore, a DDO can put itself into and out of the datastore.

One skilled in the art would recognize that these are only example classes and subclasses and other structures maybe used for objects and other classes or subclasses may be added to or removed from the tree shown in FIG. 2.

With respect to the notion of "federation", each participating datastore preserves the right to maintain its "personality", i.e. its own query language, data-model or schema, method of interaction, etc, and at the same time cooperating in a federation to provide a federated schema. This design allows the users to preserve the natural view to their favorite datastore as well as access them in conjunction with other datastores in a federated context.

The federated datastore 100 can combine the participating native datastores in two ways:

*With mapping.* As described above, mapping of concepts across participating datastores is established to provide a unified conceptual view. Based on this federated schema, federated queries with both join and union expressions can be formulated.

*Without mapping.* In this case, the federated datastore 100 only reflects the union of each participating datastore's conceptual view. Although it coordinates query processing and data-access for each underlying datastore, the federated datastore 100 must accept queries in each datastore's native language since the query translation process can not be performed without mapping. In addition, since there is no conceptual mapping between datastores, the FederatedQuery 19 results can only reflect the union of results from each datastore.

5 The embodiment of the invention is incorporated into one or more software programs that  
reside at the federated datastore 100. Generally, the software programs and the instructions  
derived therefrom, are all tangibly embodied in a computer-readable medium, e.g. one or more of  
the data storage devices, which may be connected to the federated datastore 100. Moreover, the  
software programs and the instructions derived therefrom, are all comprised of instructions which,  
when read and executed by the computer system 100, causes the computer system 100 to perform  
the steps necessary to implement and/or use the present invention. Under control of an operating  
system, the software programs and the instructions derived therefrom, may be loaded from the  
data storage devices into a memory of the federated datastore 100 for use during actual  
10 operations.

15 Thus, the present invention may be implemented as a method, apparatus, or article of  
manufacture using standard programming and/or engineering techniques to produce software,  
firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively,  
"computer program product") as used herein is intended to encompass a computer program accessible  
from any computer-readable device, carrier, or media. Of course, those skilled in the art will recognize  
many modifications may be made to this configuration without departing from the scope of the present  
invention.

20 Those skilled in the art will recognize that the exemplary environment illustrated in FIG. 1 is  
not intended to limit the present invention. Indeed, those skilled in the art will recognize that other  
alternative hardware environments may be used without departing from the scope of the present  
invention.

### Federated Datastore

FIG. 3 is a diagram illustrating a class hierarchy for Datastore classes. A main datastore class  
300 is an abstract base class (i.e., superclass) for all datastores. In particular, some datastore classes  
that are based on the datastore class 300 and inherit its characteristics are the following: a DL  
Datastore class 302, a VisualInfo Datastore class 304, a Federated Datastore class 306, and an  
OnDemand Datastore class 308. It is to be understood that the techniques of the invention may be  
applied to any data source and is not limited to the mentioned datastores.



FIG. 4 is a diagram illustrating one composition of a federated datastore. The federated datastore 400 connects to heterogeneous datastores 402, 404, 406, and 408. As illustrated, a federated datastore 406 may connect to and be nested under federated datastore 400. Additionally, the federated datastore 406 may connect to heterogeneous datastores 410, 412, and 414. The depicted architecture is only a sample, and one skilled in the art would recognize that other examples fall within the scope of the invention.

In the preferred embodiment, the federated datastore 100 takes query strings expressed in a federated query language. An example class definition for DatastoreFederated 100 is set forth below.

DKDatastoreFed.java

```
10      package com.ibm.mm.sdk.server;

      public class DKDatastoreFed extends dkAbstractDataStore
          implements DKConstantFed,
                      DKConstant,
                      DKMessageIdFed,
15      DKMessageId,
                      dkFederation,
                      java.io.Serializable
      {
          public dkCollection listEntities() throws DKException, Exception
20      public String[] listEntityNames() throws DKException, Exception
          public String[] listTextEntityNames() throws DKException, Exception
          public String[] listParmEntityNames() throws DKException, Exception
          public dkCollection listEntityAttrs(String entityName) throws DKException, Exception
          public String[] listEntityAttrNames(String entityName) throws DKException, Exception
25      public String registerMapping(DKNVPair sourceMap) throws DKException, Exception
          public void unregisterMapping(String mappingName) throws DKException, Exception
          public String[] listMappingNames() throws DKException, Exception
```

```

public dkSchemaMapping getMapping(String mappingName) throws DKException,
    Exception
public synchronized dkExtension getExtension(String extensionName) throws
    DKException, Exception
5 public synchronized void addExtension(String extensionName,
    dkExtension extensionObj) throws DKException, Exception
public synchronized void removeExtension(String extensionName) throws
    DKException, Exception
public synchronized String[] listExtensionNames() throws DKException, Exception
10 public DKDDO createDDO(String objectType,
    int Flags) throws DKException, Exception
public dkCollection listSearchTemplates() throws DKException, Exception
public String[] listSearchTemplateName() throws DKException, Exception
public dkSearchTemplate getSearchTemplate(String templateName) throws
15 DKException, Exception
public void destroy() throws DKException, Exception
public synchronized string addRemoveCursor (dkResultSetCursor iCurt int action)
    throws DKException, Exception
public dkDatastore datastoreByServerName (String dsType, String dsName)
20 throws DKException, Exception
public void changePassword (String serverName,
    String user Id,
    String oldPwd,
    String newPwd)
25 throws DKException, Exception
public void requestConnection (String serverName,
    String userId,
    String passwd,
    String connectString)
30 throws DKException, Exception

```

```

public void excludeServer (String serverName, String templateName)
    throws DKException, Exception
public boolean isServerExcluded (String serverName, String templateName)
    throws DKException, Exception, java.rmi.RemoteException
5  public String[] listExcludedServers(String templateName) throws DKException,
    Exception
    public void clearExcludedServers(String templateName) throws DKException,
    Exception
};

```

10 The following methods are part of the federated datastore class:

*public DKDatastoreFed() throws DKException, Exception*

Constructs default Federated Datastore.

*public DKDatastoreFed(String configuration) throws DKException, Exception*

Constructs default Federated Datastore.

15 *public void connect(String datastore\_name,*  
*String user\_name,*  
*String authentication,*  
*String connect\_string) throws DKException, Exception*

Establishes a connection to a federated datastore.

20 Parameters:

datastore\_name - federated datastore name  
 user\_name - userid to logon to this federated datastore  
 authentication - password for this user\_name

connect\_string - additional information string

Throws: DKException

if either:

datastore\_name, user\_name, or authentication is null

or if error occurs in the federated datastore

Overrides:

connect in class dkAbstractDatastore

*public void disconnect() throws DKException, Exception*

Disconnects from the federated datastore.

Throws: DKException

if unable to disconnect from server.

Overrides:

disconnect in class dkAbstractDatastore

*public Object getOption(int option) throws DKException*

Gets defined datastore option

Parameters:

option - an option id

Returns:

the value for the given option

Throws: DKException

if option is not set

Overrides:

getOption in class dkAbstractDatastore

*public void setOption(int option, Object value) throws DKException*

Sets the given "option" with a specific "value".

Parameters:

- 5                    option - an option id
- value - the value for the "option"

Throws: DKException

if option/value is invalid

Overrides:

- 10                    setOption in class dkAbstractDatastore

*public Object evaluate(String command,  
                          short commandLangType,  
                          DKNVPair params[]) throws DKException, Exception*

Evaluates a query and returns the result as a dkQueryableCollection object.

15                    Parameters:

- command - a query string that represent the query criteria
- commandLangType - a query language type, for Federated, it will be  
                                        DK\_FEDERATED\_QL\_TYPE
- params - a name/value pairs list

20                    Returns:

a query result collection

Throws: DKException

if "command" argument is null

Overrides:

- 25                    evaluate in class dkAbstractDatastore

*public Object evaluate(dkQuery query) throws DKException, Exception*

Evaluates a query and returns the result as a dkQueryableCollection.

Parameters:

5                      query - a given query object

Returns:

a query result collection

Throws: DKException

if the "query" input is null or not of federated query type.

10              Overrides:

evaluate in class dkAbstractDatastore

*public Object evaluate(DKCQExpr qe) throws DKException, Exception*

Evaluates a query.

Parameters:

15                      qe - a common query expression object

Returns:

a collection of the results

Throws: DKException

if common query expression object is invalid

20              Overrides:

evaluate in class dkAbstractDatastore

*public dkResultSetCursor execute(String command,  
                                         short commandLangType,  
                                         DKNVPair params[]) throws DKException, Exception*

Executes a command query of the federated datastore and returns a result set cursor.

Parameters:

command - a query string that represents the query criteria.

commandLangType - a query language type, for Federated, it will be

5 DK\_FEDERATED\_QL\_TYPE.

params[] - a name/value pairs list.

Returns:

a dkResultSetCursor object.

Throws: DKException

10 if "command" is null or invalid, or "commandLangType" is not Federated  
Query type.

Overrides:

execute in class dkAbstractDatastore

*public dkResultSetCursor execute(dkQuery query) throws DKException, Exception*

15 Executes a command query of the federated datastore and returns a result set cursor. This  
method takes a Federated query object as an argument.

Parameters:

query - a federated dkQuery object

20 Returns:

a dkResultSetCursor object

Throws: DKException

if "query" object is null or query.qfType() is not

DK\_FEDERATED\_QL\_TYPE

25 Overrides:

execute in class dkAbstractDatastore

*public dkResultSetCursor execute(DKCQExpr cqe) throws DKException, Exception*

Executes a query expression.

Parameters:

5                   cqe - a common query expression object

Returns:

resultSetCursor which represents a federated datastore cursor.

Throws: DKException

if "cqe" object is invalid

10               Overrides:

execute in class dkAbstractDatastore

*public void executeWithCallback(dkQuery query,  
                                  dkCallback callbackObj) throws DKException, Exception*

Executes a query with callback function.

15               Parameters:

query - a query object

callbackObj - a dkCallback object

Overrides:

executeWithCallback in class dkAbstractDatastore

20   *public void executeWithCallback(String command,  
                                  short commandLangType,  
                                  DKNVPair params[],  
                                  dkCallback callbackObj) throws DKException, Exception*

Execute the query with callback function.



Parameters:

command - a query string  
commandLang - a query type  
params - additional query option in name/value pair  
callbackObj - a dkCallback object

Overrides:

executeWithCallback in class dkAbstractDatastore

*public void executeWithCallback(DKCQExpr cqe,  
dkCallback callbackObj) throws DKException, Exception*

10 Execute a query expression with callback function.

Parameters:

cqe - a common query expression object  
callbackObj - a dkCallback object

Overrides:

15 executeWithCallback in class dkAbstractDatastore

*public dkQuery createQuery(String command,  
short commandLangType,  
DKNVPair params[]) throws DKException*

Creates a federated query object.

20 Parameters:

command - a query string that represents the query criteria  
commandLangType - a query language type, it will be one of the  
following:

DK\_CM\_TEMPLATE\_QL\_TYPE

DK\_CM\_TEXT\_QL\_TYPE  
DK\_CM\_IMAGE\_QL\_TYPE  
DK\_CM\_PARAMETRIC\_QL\_TYPE  
DK\_CM\_COMBINED\_QL\_TYPE

5                   params[] - a name/value pairs list

Returns:

a federated dkQuery object

Throws: DKException

if "command" is null

10                 Overrides:

createQuery in class dkAbstractDatastore

*public dkQuery createQuery(DKCQExpr qe) throws DKException*

Creates a query object.

15                 Parameters:

cqe - a common query expression object

Throws: DKException

if "cqe" object is invalid

Overrides:

20                 createQuery in class dkAbstractDatastore

*public dkCollection listDataSources() throws DKException*

List the available datastore sources that a user can connect to.

Returns:

a collection of ServerDef objects describing the servers

Throws: DKException

if internal error occurs from server

Overrides:

listDataSources in class dkAbstractDatastore

5 *public String[] listDataSourceNames() throws DKException*

Gets a list of datasource names.

Returns:

an array of datasource names

Throws: DKException

10 if error occurs when retrieving datasource names

Overrides:

listDataSourceNames in class dkAbstractDatastore

*public void addObject(dkDataObject dataobj) throws DKException, Exception*

Adds a DDO object.

15 Parameters:

ddo - a Federated object to be added.

Throws: DKException

if error occurs during add.

Overrides:

20 addObject in class dkAbstractDatastore

*public void deleteObject(dkDataObject dataobj) throws DKException, Exception*

Deletes a data object.

Parameters:

ddo - a federated DDO object to be deleted

Throws: DKException

5 if error occurs during delete.

Overrides:

deleteObject in class dkAbstractDatastore

*public void retrieveObject(dkDataObject dataobj) throws DKException, Exception*

10 Retrieves a data-object.

Parameters:

ddo - document object to be retrieved.

Throws: DKException

when retrieve failed.

15 Overrides:

retrieveObject in class dkAbstractDatastore

*public void updateObject(dkDataObject dataobj) throws DKException, Exception*

Updates a data-object.

20 Parameters:

ddo - the data-object to be updated.

Throws: DKException

if error occurs in the datastore

Overrides:

25 updateObject in class dkAbstractDatastore

*public void commit() throws DKException*

Commits all activities since the last commit.

Throws: DKException

is thrown since federated datastore does not support transaction scope for now.

5

Overrides:

commit in class dkAbstractDatastore

*public void rollback() throws DKException*

Rolls back all activities since the last commit.

Throws: DKException

10

is thrown since Federated does not support transaction scope for now.

Overrides:

rollback in class dkAbstractDatastore

*public boolean isConnected()*

Checks to see if the datastore is connected

15

Returns:

true if connected, false otherwise

Overrides:

isConnected in class dkAbstractDatastore

*public DKHandle connection() throws Exception*

20

Gets the connection handle for the datastore.

Returns:

the connection handle

Overrides:

connection in class dkAbstractDatastore

5 *public DKHandle handle(String type) throws Exception*

Gets a datastore handle.

Parameters:

type - type of datastore handle wanted

Returns:

10 a datastore handle

Overrides:

handle in class dkAbstractDatastore

*public String userName()*

Gets the user name that user used to logon to the datastore.

15 Returns:

the userid that user used to logon

Overrides:

userName in class dkAbstractDatastore

*public String datastoreName() throws Exception*

20 Gets the name of this datastore object. Usually it represents a datastore source's server name.

Returns:

datastore name

Overrides:

datastoreName in class dkAbstractDatastore

5 *public String datastoreType() throws Exception*

Gets the datastore type for this datastore object.

Returns:

datastore type

Overrides:

10 datastoreType in class dkAbstractDatastore

*public dkDatastoreDef datastoreDef() throws DKException, Exception*

Gets datastore definition.

Returns:

the meta-data (dkDatastoreDef) of this datastore

15 Overrides:

datastoreDef in class dkAbstractDatastore

*public dkCollection listEntities() throws DKException, Exception*

Gets a list of federated entities from Federated server.

20 Returns:

a collection of dkEntityDef

Throws: DKException

if error occurs

Overrides:

listEntities in class dkAbstractDatastore

*public String[] listEntityNames() throws DKException, Exception*

5 Gets a list of federated entities names from Federated server.

Returns:

an array of names

Throws: DKException

if error occurs

10 Overrides:

listEntityNames in class dkAbstractDatastore

*public String[] listTextEntityNames() throws DKException, Exception*

Gets a list of federated text search entities names from Federated server.

Returns:

15 an array of names

Throws: DKException

if error occurs

*public String[] listParmEntityNames() throws DKException, Exception*

Gets a list of federated parametric search entities names from Federated server.

20 Returns:

an array of names



Throws: DKException  
if error occurs

Overrides:  
listEntityAttrs

5 *public dkCollection listEntityAttrs(String entityName) throws DKException, Exception*

Gets a list of attributes for a given entity name.

Parameters:  
entityName - name of entity to retrieve attributes for

Returns:  
10 a dkCollection of dkAttrDef objects

Throws: DKException  
if the entity name does not exist

Overrides:  
listEntityAttrs in class dkAbstractDatastore

15 *public String[] listEntityAttrNames(String entityName) throws DKException, Exception*

Gets a list of attribute names for a given entity name.

Parameters:  
entityName - name of entity to retrieve attribute names for

Returns:  
20 an array of attribute names

Throws: DKException  
if the entity name does not exist

Overrides:  
listEntityAttrNames in class dkAbstractDatastore

*public String registerMapping(DKNVPair sourceMap) throws DKException, Exception*

Registers a mapping definition to this datastore. Mapping is done by entities.

Parameters:

5                   sourceMap - source name and mapping, a DKNVPair class with the following possible values:

                  ("BUFFER", ) : buffer\_ref is a reference to a string in memory

                  ("FILE", ) : file\_name is the name of the file containing the mapping

10                   ("URL", ) : URL-address location of the mapping

                  ("LDAP", ) : LDAP file-name

                  ("SCHEMA", ) : a reference to a dkSchemaMapping object

defining the

                  mapping. Currently, only "SCHEMA" option is supported, others

15                   may be

                  added later.

Returns:

                  the name of the mapping definition.

Overrides:

20                   registerMapping in class dkAbstractDatastore

See Also:

                  unRegisterMapping

*public void unRegisterMapping(String mappingName) throws DKException, Exception*

Unregisters mapping information from this datastore.

25                   Parameters:

                  mappingName - name of the mapping information

Overrides:

unRegisterMapping in class dkAbstractDatastore

See Also:

registerMapping

5 *public String[] listMappingNames() throws DKException, Exception*

Gets the list of the registered mappings for this datastore.

Returns:

an array of registered mapping objects' names. The array length would be zero if there is no mapping registered.

10

Overrides:

listMappingNames in class dkAbstractDatastore

See Also:

registerMapping

*public dkSchemaMapping getMapping(String mappingName) throws DKException, Exception*

15

Gets mapping information from this datastore.

Parameters:

mappingName - name of the mapping information

Returns:

the schema mapping object

20

Overrides:

getMapping in class dkAbstractDatastore

See Also:

registerMapping

*public synchronized dkExtension getExtension(String extensionName) throws DKException,  
Exception*

Gets the extension object from a given extension name.

Parameters:

5                      extensionName - name of the extension object.

Returns:

extension object.

Overrides:

getExtension in class dkAbstractDatastore

10    *public synchronized void addExtension(String extensionName,  
                                                 dkExtension extensionObj) throws DKException, Exception*

Adds a new extension object.

Parameters:

extensionName - name of new extension object

15                      extensionObj - the extension object to be set

Overrides:

addExtension in class dkAbstractDatastore

*public synchronized void removeExtension(String extensionName) throws DKException,  
Exception*

20                      Removes an existing extension object.

Parameters:

extensionName - name of extension object to be removed

Overrides:

removeExtension in class dkAbstractDatastore

*public synchronized String[] listExtensionNames() throws DKException, Exception*

Gets the list of extension objects' names.

5 Returns:

an array of extension objects' names

Overrides:

listExtensionNames in class dkAbstractDatastore

10 *public DKDDO createDDO(String objectType,  
int Flags) throws DKException, Exception*

Creates a new DDO with object type, properties and attributes set for a given back-end server.

Parameters:

objectType - the object type

15 Flags - to indicate various options and to specify more detailed characteristics of the DDO to create. For example, it may be a directive to create a document DDO, a folder, etc.

Returns:

20 a new DDO of the given object type with all the properties and attributes set, so that the user only needs to set the attribute values

Overrides:

createDDO in class dkAbstractDatastore

*public dkCollection listSearchTemplates() throws DKException, Exception*

Gets a list search templates from a federated server.

Returns:

a DKSequentialCollection of search templates

Throws: DKException

5 if internal datastore error occurs

*public String[] listSearchTemplateName() throws DKException, Exception*

Gets a list search templates' names from a federated server.

Returns:

an array of search template names

10 Throws: DKException

if internal datastore error occurs

*public dkSearchTemplate getSearchTemplate(String templateName) throws DKException, Exception*

Gets a search template information from a given template name.

15 Returns:

dkSearchTemplate object.

Throws: DKException

if internal datastore error occurs

*public void destroy() throws DKException, Exception*

20 datastore destroy - datastore cleanup if needed

Overrides:

destroy in class dkAbstractDatastore

*public synchronized string addRemoveCursor (dkResultSetCursor iCurt int action)  
throws DKException, Exception*

5 *public dkDatastore datastoreByServerName (String dsType, String dsName)  
throws DKException, Exception*

Gets a reference to the specified datastore. The datastore must be connected, otherwise it will return null even if one is found. First, it will look in the free connection pool. If none found, it will look under the connection pool held by active cursors.

10 *public void changePassword (String serverName,  
String user Id,  
String oldPwd,  
String newPwd)  
throws DKException, Exception*

15 Changes the password of a given user Id for a specified server. Administrator only function.

Parameters:

20           userId     - the user-id  
            oldPwd    - the old password  
            newPwd    -the new password

*public void requestConnection (String serverName,  
String userId,  
String passwd,  
25 String connectString)  
throws DKException, Exception*

Requests a connection to a particular server with the given userid, password & connectString.

Parameters:

30           userId     -the user Id  
            passwd    -the password

connectString - the connect string to logon

*public void excludeServer (String serverName, String templateName)  
throws DKException, Exception*

Requests the named server to be skipped for the named search template.

5       Parameters:  
          serverName - a back end server name  
          templateName - a search template name

*public boolean isServerExcluded (String serverName, String templateName)  
throws DKException, Exception, java.rmi.RemoteException*

10       Checks if the given server is in the excluded list for the named search template.

Parameters:  
          serverName       - a back end server name  
          templateName - a search template name

15       Returns:  
          true or false

*public String[] listExcludedServers(String templateName) throws DKException, Exception*

Lists all the excluded servers for the named search template

20       Parameters:  
          s - templateName - a search template name  
Returns:  
          an array of server names that were excluded during search

*public void clearExcludedServers(String templateName) throws DKException, Exception*

Clears all the excluded servers for the named search template

25       Parameters:  
          s - templateName - a search template name

The following is sample syntax of a federated query string. However, it is to be understood that other syntax, including other parameters, may be used for the federated query string without departing from the scope of the invention.



```

PARAMETRIC_SEARCH=([ENTITY=entity_name,]
                    [MAX_RESULTS=maximum_results,]
                    [COND=(conditional_expression)]
                    [; ...]
5                );
                [OPTION=([CONTENT=yes_no]
                        )]
                [and_or

TEXT_SEARCH=(COND=(text_search_expression)
10                );
                [OPTION=([SEARCH_INDEX={search_index_name | (index_list) };]
                        [MAX_RESULTS=maximum_results;]
                        [TIME_LIMIT=time_limit]
                        )]
15                ]

                [and_or

IMAGE_SEARCH=(COND=(image_search_expression)
                );
                [OPTION=([SEARCH_INDEX={search_index_name | (index_list) };]
20                        [MAX_RESULTS=maximum_results;]
                        [TIME_LIMIT=time_limit]
                        )]

                ]

```

25 There are several mechanisms for users to submit federated queries for execution. For example, users can create a federated query string and pass it to a federated query object and then invoke an execute or evaluate method on that object to trigger the query processing. Alternatively, a user can pass the federated query string to the execute or evaluate method in the federated datastore to process the query directly. The query string will be parsed into a federated query canonical form (query expression), which is essentially a datastore neutral representation of the query. In case the

30 input query comes from a graphical user interface (GUI) based application, the query does not need to be parsed and the corresponding canonical form can be directly constructed.

The query canonical form is the input for the federated query processor module. This module will perform the following tasks:

*Query translation.* Translates the query canonical form into several native queries that corresponds to each native datastore associated to this federated datastore . The translation information is obtained from the schema mapping.

5        *Data conversion.* Converts data in the query into a native data type for each of the associated native datastores. This process uses the mapping and conversion mechanisms described in the schema mapping.

*Data filtering.* Filters only the relevant data during the construction of native queries.

Each native query is submitted to the corresponding native datastore for execution. Initially, the results returned are cursors to the data in each datastore.

10        The end-result of an initial query is a federated result set cursor object, which is a virtual collection (i.e., at this time, data has not actually been retrieved) of cursors to objects in each of the native datastores.

15        The user can retrieve the actual data using a fetch. When a fetch is issued for data, the data is returned by the native datastores to the federated query results processor module, which will do the following:

*Data conversion.* Converts data from the native type into a federated type according to the mapping information.

*Data filtering.* Filters the results to include only the requested data.

*Result merging.* Merges the results from several native datastores into a federated collection.

20        The federated result set cursor object provides the facility to separate query results according to the source native datastores. To do such a processing, the user/application may either use the federated cursor to fetch data or a native datastore cursor to fetch data from a particular datastore.

A FederatedQuery represents and executes queries across heterogeneous datastores. This query can be a combination of a DL parametric query, OnDemand query, and other query types involving supported datastores. To retrieve data from each datastore, the federated datastore delegates the query processing task to each of the native datastores.

## 5 DKFederatedQuery.java

```
package com.ibm.mm.sdk.common.DKFederatedQuery
```

```
public class DKFederatedQuery
```

```
    extends Object
```

```
    implements dkQuery, DKConstant, DKMessageId, Serializable
```

```

10    {
        public DKFederatedQuery(dkDatastore creator,
                                String queryString)
        public DKFederatedQuery(dkDatastore creator,
        public DKFederatedQuery(DKFederatedQuery fromQuery)
15    public void prepare(DKNVPair params[]) throws DKException, Exception
        public void execute(DKNVPair params[]) throws DKException, Exception
        public int status()
        public Object result() throws DKException, Exception
        public dkResultSetCursor resultSetCursor() throws DKException, Exception
20    public short qlType()
        public String queryString()
        public dkDatastore getDatastore()
        public void setDatastore(dkDatastore ds) throws DKException, Exception
        public String getName()
25    public void setName(String name)
        public int numberOfResults()
    };

```

The following methods are part of the federated query class:

```
public DKFederatedQuery(dkDatastore creator,  
    String queryString)
```

5 Constructs a Federated query.

Parameters:

creator - datastore

queryString - a query string

```
public DKFederatedQuery(dkDatastore creator,  
    DKCQExpr queryExpr)
```

10

Constructs a Federated query

Parameters:

creator - datastore

queryExpr - a query expression

```
15 public DKFederatedQuery(DKFederatedQuery fromQuery)
```

Constructs a Federated query from a Federated query object.

Parameters:

fromQuery - Federated query

```
20 public void prepare(DKNVPair params[]) throws DKException, Exception
```

Prepares a query.

Parameters:

params - additional prepare query option in name/value pair

*public void execute(DKNVPair params[]) throws DKException, Exception*

5 Executes a query.

Parameters:

params - additional query option in name/value pair

*public int status()*

Gets query status.

10 Returns:  
query status

*public Object result() throws DKException, Exception*

Gets query result.

Returns:

15 query result in a DKResults object

*public dkResultSetCursor resultSetCursor() throws DKException, Exception*

Gets query result.

Returns:

query result in a dkResultSetCursor object

*public short qlType()*

Gets query type.

5

Returns:

query type

*public String queryString()*

Gets query string

Returns:

10

query string

*public dkDatastore getDatastore()*

Gets the reference to the owner datastore object.

Returns:

the dkDatastore object

15 *public void setDatastore(dkDatastore ds) throws DKException, Exception*

Sets the reference to the owner datastore object.

Parameters:

ds - a datastore

*public String getName()*

Gets query name.

Returns:

name of this query

5 *public void setName(String name)*

Sets query name.

Parameters:

name - new name to be set to this query object

*public int numberOfResults()*

10 Gets the number of query results.

Returns:

number of query results

### Schema Mapping

15 A schema mapping represents a mapping between the schema in a datastore with the structure of the data-object that the user wants to process in memory. Schema mapping has been generally described in U.S. Patent Application Nos. 08/276,382 and 08/276,747, also assigned to IBM.

20 A federated schema is the conceptual schema of a federated datastore 100, which defines a mapping between the concepts in the federated datastore 100 to concepts expressed in each participating datastore schema. In general, a schema mapping handles the difference between how the data are stored in the datastore (as expressed by the datastore's conceptual schema) and how the

user wants to process them in the application program. This mapping can also be extended to incorporate relationship associations among entities in a federated datastore, e.g., associating an employee's name with the appropriate department name. Since the mapping process can be a bit tedious, it is usually done with the help of a typical GUI-oriented schema mapping program.

- 5           In addition to schema-mapping information involving the mapping of entities and attributes, a federated datastore 100 must also have access to the following information:

*User-id and password mapping.* To support single sign-on features, each user-id in the federated datastore 100 needs to be mapped to its corresponding user-ids in the native datastores.

- 10           *Datastore registration.* Each native datastore needs to be registered so it can be located and logged-on to by the federated datastore 100 processes on behalf of its users.

An Architecture and Implementation of a Dynamic RMI Server Configuration Hierarchy to Support Federated Search and Update Across Heterogeneous Datastores

- 15           An embodiment of the invention provides an architecture and implementation of a dynamic RMI server configuration hierarchy ("RMI architecture") to support federated search and update across heterogeneous datastores. In particular, the RMI architecture enables addition and deletion of RMI servers dynamically. Additionally, the RMI architecture provides load balancing among the RMI servers. Federated systems may be connected to the RMI servers.

- 20           The RMI architecture supports a hierarchical grouping of servers on the same or different machines. With the hierarchical grouping, the RMI architecture supports search and update of heterogeneous datastores participating in a federated system, within a client/server environment. In one embodiment, the RMI servers are Java RMI servers. RMI stands for Remote Method Invocation, which identifies a set of protocols developed by Sun Microsystems. The protocols enable Java objects
- 25           to communicate remotely with other Java objects.

FIG. 5 is a diagram of an extended Grand Portal architecture. A Grand Portal client for a federated client datastore 500 is connected to a Grand Portal server for a federated server datastore 502. Another federated client/server system 504 may be connected to the federated server 502. A Grand



Portal client/server system for an OnDemand (OD) datastore 506 may be part of the federation. Additionally, a Grand Portal client/server system for a Digital Library/VisualInfo (DL/VI) datastore 508 may be part of the federation. As with any of the datastores discussed herein, a user may access the client or the server directly. Therefore, user applications may reside at either the client or the server.

A Grand Portal client for a DES datastore 510 or a Grand Portal server for a DES datastore 512 may each be connected to the federation. While the DL/VI datastore enables searching a DL/VI Library server and the OD datastore enables searching of an OnDemand datastore, the DES datastore enables searching of multiple other datastores. In particular, the DES datastore enables searching of a Lotus Notes server 514, a Web 516, a file system 518, and a relational database 520.

FIG. 6 is a diagram illustrating individual datastores and federated compositions. In particular, a datastore can be configured as a stand-alone or as part of a federation. Additionally, a federated datastore can be composed of any number of datastores, including other federated datastores. Stand-alone datastores may be accessed directly by a user. The following are example stand-alone datastores in FIG. 6: a Digital Library (DL) datastore 600, an OnDemand datastore 602, a VisualInfo/400 datastore 604, a Domino.Doc datastore 606, or a ImagePlus/390 datastore 608. Additionally, a DES datastore 610 maybe a stand alone in that it is not part of a federated composition. A federated composition 612 may include individual datastores 614 and 616, another federated datastore 618, and a search gateway to a DES datastore 620. In turn, the DES datastore 620 enables searching a Lotus Notes database 622, searching the Web 624, searching a file system 626, or searching a relational database 628 (e.g., DB2, Oracle, or ODBC).

The RMI architecture allows a user to configure a hierarchical grouping of RMI servers either on several different machines or on the same machine to support federated search and update across several heterogeneous datastores. The architecture allows the creation of a flexible tree of RMI servers in which a new server can be attached or removed dynamically from the configuration. This feature is very advantageous for a federated search environment in a client/server setting where the configuration, the number, and the type of datastores participating in the federation changes dynamically over time.

FIG. 7 is a diagram illustrating Remote Method invocation (RMI) client/server hierarchy. An RMI server can connect to an infinite number of datastores, but each server must be connected to at

least one datastore. The master RMI server (A) 700 can reference sub-RMI (B) servers 702 and 704 that reside below in the hierarchy. Additionally, another sub-RMI (C) server 706 may be below sub-RMI (B) servers 702 and 704 in the hierarchy.

If an RMI client is searching for the first time for a datastore, the search begins with RMI server (A) 700. If the datastore is not found in RMI server (A) 700, the sub-RMI servers (B and then C) are searched next. If the same RMI client searches for the datastore again, the client searches in the RMI server (A or B or C) where it found the datastore the first time.

That is, the RMI server configuration is initially constructed with a single server supporting one or more federated datastores and/or one or more native datastores. For example, a federated configuration (consisting of a federated datastore and one or more native datastores) and a Digital Library datastore may be connected to a single RMI server. This RMI server is at the top of the RMI server hierarchy and serves as a primary node. Via the RMI server, either the federated configuration or the stand alone Digital Library datastore may be searched.

When a new RMI server is needed, it can be configured in a different machine. This additional machine registers or attaches itself to an existing server in the RMI server hierarchy. In a federated search environment, a text search server (e.g., TextMiner) can be defined and attached to a Digital Library server by specifying its host name and port number. Additional servers can be defined at the same or different machines and attached either to the primary node or any node below it. For example, an image search server QBIC (i.e., Query by Image Content), VisualInfo/400, Image Plus/390, DB2, OnDemand, etc. could be attached to an existing server.

Each RMI server is defined with a server type and a maximum number of connections that it can handle. This information is used by the RMI architecture to perform load balancing and to distribute loads among several servers. The load balancing technique is based on the percentage of the current load and the maximum load of the server. For example, if there are two servers (ServerA and ServerB), assume that ServerA can take 5 loads, while ServerB can take 100 loads. If ServerA is handling 4 loads, while ServerB is handling 20 loads, ServerA is handling a larger percentage of loads for its capability. Therefore, when another request for data is received, ServerB is selected to process the request.

When a federated search request is submitted by a user, the federated datastore will consult the primary node to locate the server with the proper type and allowable loads. Then, the federated

datastore will direct the search request to the selected server. Once the server capable of providing the requested service is located, any subsequent requests are automatically directed to the selected server, transparent from the user. This situation is depicted in FIG. 7 by the dotted arrow.

FIG. 8 is a flow diagram of one use of the RMI architecture. In block 800, one or more master RMI servers are initially configured. In block 802, for each master RMI server, upon receiving a request to add another RMI server, the new RMI server is dynamically connected to an existing server within the RMI server hierarchy based on factors, including the number of connections available at the existing server. In block 804, upon receiving a request to delete a RMI server, the RMI server is dynamically deleted from the RMI server hierarchy.

FIG. 9 is a flow diagram illustrating searching within a RMI server hierarchy. Initially, in block 900, a federated datastore receives a request for data from a user or application program. In block 902, the federated datastore identifies RMI servers that are attached to data sources that can satisfy the request for data. In block 904, the federated server selects an identified RMI server based on its current load of search requests. In block 906, the federated datastore selects an identified RMI server based on its current load of search requests, and then the federated datastore forwards the request for data to the selected RMI server. The data source at the selected RMI server processes the request for data. Additionally, in block 908, the federated datastore routes additional requests for that type of data from the same user or application program to that selected RMI server.

When Remote Method Invocation (RMI) is used with content servers, because the client classes in the Java API need to communicate with the server classes to access and manipulate data through the network, both the server and client side should be ready for client/server execution. On the server side, daemon (i.e., a process that runs in the background and performs a particular operation at a specified task or based on a specified event) should be ready to receive a request from a client using a specified port number. On the client side, an application program requires a server name and port number. To communicate between client and server, the port number of the client and server must be same.

To start a daemon on the server side, a script file is used. For example, on Windows NT, a script file called cmbregist.cmd is used, and AIX, a script file called cmbregist.sh is used. Before starting the daemon, the correct port number and server type are defined.

The following table describes the type code used for each server type.

Server Type	Server Type Code
Digital Library and VisualInfo	DL
VisualInfo for AS/400	V4
ImagePlus for OS/390	IP
OnDemand	OD
Domino.Doc	DD
Domino Extended Search	DES
Federated Datastore	Fed

5

The cmbregist script file starts a daemon for the RMI server (A) 700, in FIG. 7, in Windows

10 NT. The following is an example cmbregist script file.

```
set remotePort=<port#>
jre -cp %CLASSPATH% -ms16M \
-Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
%remotePort% <max# of connections> <#of servers> <list of server types>
```

15 echo "Regist is over"

In the script file, the parameters <port#>, <max # of connections>, <# of servers>, and <server types> are replaced with values. Also, note that there is a space between // and com. The following is an example:

```
set remotePort=1919
jre -cp %CLASSPATH% -ms16M \
-Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
1919 0 3 DD IP DL
```

20

echo "Regist is over"

If <max # of connections> is zero, then an infinite number of connections can be established to the RMI server from the client.

A different version of cmbregist is used to run a daemon on the server side. In particular, the following sample script starts a daemon for a sub-RMI server (B or C) 702, 704, or 705, in FIG. 7, in Windows NT:

```
set remotePort=<port#>
jre -cp %CLASSPATH% -ms16M \
-Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
10      %remotePort% <#of connections> MasterRMIServer <hostName> <port# of
MasterRMIServer> \
      <#of servers> <list of server types>

echo "Regist is over"
```

In the script file, the parameters <port#>, <# of connections>, <hostname>, <port# of master RMI server>, <# of servers>, and <server types> are replaced with values. The following is an example:

```
set remotePort=1910
jre -cp %CLASSPATH% -ms16M \
-Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
20      1910 0 MasterRMIServer voodoo 1919 3 Fed DD DL

echo "Regist is over"
```

Note that server names are case sensitive. For example, if an OnDemand datastore is named OD in a script for the master RMI server, it is named OD in the sub-RMI server. After updating a script file, cmbregist is used to run the daemon on the server side.

For AIX, a sample cmbregist.sh script file looks like the following:

```
set remotePort=<port#>
jre -cp $CLASSPATH -ms32M \
    -Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
5    $remotePort <max# of connections> <#of servers> <list of server types> &

echo "Regist is over"
```

In the script file, the parameters <port#>, <max # of connections>, <# of servers>, and <server types> are replaced with values. The following is an example:

```
set remotePort=1919
10 jre -cp $CLASSPATH -ms32M \
    -Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
    1919 0 3 DD IP DL &

echo "Regist is over"
```

The following script will start the daemon for a sub-RMI server in AIX:

```
15 set remotePort=<port#>
jre -cp $CLASSPATH -ms32M \
    -Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
        $remotePort <#of connections> MasterRMIServer <hostName> <port# of
MasterRMIServer> \
20    <#of servers> <list of server types> &

echo "Regist is over"
```

In the script file, the parameters <port#>, <# of connections>, <hostname>, <port# of master RMI server>, <# of servers>, and <server types> are replaced with values. The following is an example:

```
set remotePort=1910
jre -cp $CLASSPATH -ms32M \
5   -Djava.rmi.server.codebase=http:// com.ibm.mm.sdk.remote.DKRemoteMainImp \
    1910 0 MasterRMIServer voodoo 1919 3 Fed DD DL &

echo "Regist is over"
```

The following statements are used to set up the main/primary server (e.g., RMI server (A) 700 in FIG. 7). A script file cmbregis.bat (NT) or script file cmbregist.sh (AIX) may be used.

```
- On NT: jre -cp %classpath% -ms16M -Djava.rmi.server.codebase=http://
com.ibm.mm.sdk.remote.DKRemoteMainImp %remotePort% 0 10 DL TS QBIC Fed
JDBC V4 IP DD OD DES

- On AIX: jre -ms32M -cp $CLASSPATH -Djava.rmi.server.codebase=http://
15 com.ibm.mm.sdk.remote.DKRemoteMainImp $remotePort 0 5 TS QBIC DL JDBC Fed
```

The following statements are used to set up secondary servers (i.e., to set up the RMI server hierarchy). To set up another RMI server that points to the RMI server above, a user specifies the following in a copied version of the cmbregist file.

- On NT: jre -cp %classpath% -ms16M -Djava.rmi.server.codebase=http://  
20 com.ibm.mm.sdk.remote.DKRemoteMainImp %remotePort% 5  
MasterRMIServer mach1 1919 1 DL
- On AIX: jre -ms32M -cp \$CLASSPATH -Djava.rmi.server.codebase=http://  
com.ibm.mm.sdk.remote.DKRemoteMainImp \$remotePort 10 MasterRMIServer  
mach2 1919 3 TS QBIC DL Fed

### Conclusion

This concludes the description of the preferred embodiment of the invention. The following describes some alternative embodiments for accomplishing the present invention. For example, any type of computer, such as a mainframe, minicomputer, personal computer, mobile device, or embedded system, or computer configuration, such as a timesharing mainframe, local area network, or standalone personal computer, could be used with the techniques of the present invention.

The foregoing description of the preferred embodiment of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto.



WHAT IS CLAIMED IS:

1           1.       A method of searching for data in one or more heterogeneous data sources within a  
2 computer system, the method comprising the steps of:  
3           receiving a request for data at a federated data source; and  
4           selecting a server to process the request based on a load of the server and based on whether the  
5 server can satisfy the request for data.

1           2.       The method of claim 1, further comprising forwarding the request to the selected  
2 server.

1           3.       The method of claim 2, further comprising forwarding additional requests for similar  
2 data to the selected server.

1           4.       The method of claim 1, wherein the server is within a server hierarchy.

1           5.       The method of claim 1, further comprising, upon receiving a request to add another  
2 server, connecting the server to an existing server in the server hierarchy based on a number of  
3 connections of the existing server.

1           6.       The method of claim 1, further comprising, upon receiving a request to delete an  
2 existing server in the hierarchy, deleting that server.

1           7.       An apparatus for searching for data in one or more heterogeneous data sources,  
2 comprising:  
3 a computer system having one or more heterogeneous data sources; and  
1 one or more computer programs, performed by the computer system, for receiving a request  
2 for data at a federated data source and selecting a server to process the request based on a load of the  
3 server and based on whether the server can satisfy the request for data.

1           8.       The apparatus of claim 7, further comprising forwarding the request to the selected  
2   server.

1           9.       The apparatus of claim 8, further comprising forwarding additional requests for similar  
2   data to the selected server.

1            10.     The apparatus of claim 7, wherein the server is within a server hierarchy.

1            11.     The apparatus of claim 7, further comprising, upon receiving a request to add another  
2     server, connecting the server to an existing server in the server hierarchy based on a number of  
3     connections of the existing server.

1           12.     The apparatus of claim 7, further comprising, upon receiving a request to delete an  
2     existing server in the hierarchy, deleting that server.

1           13.     An article of manufacture comprising a program storage medium readable by a  
2     computer system and embodying one or more instructions executable by the computer system to  
3     perform method steps for searching for data in one or more heterogeneous data sources within a  
4     computer system, the method comprising the steps of:

1 receiving a request for data at a federated data source; and  
2 selecting a server to process the request based on a load of the server and based on whether the  
3 server can satisfy the request for data.

1           14.     The article of manufacture of claim 13, further comprising forwarding the request to  
2     the selected server.

1           15.     The article of manufacture of claim 14, further comprising forwarding additional  
2     requests for similar data to the selected server.

1            16.     The article of manufacture of claim 13, wherein the server is within a server hierarchy.

09399696 "093499  
667260" 9696660

1           17.     The article of manufacture of claim 13, further comprising, upon receiving a request  
2     to add another server, connecting the server to an existing server in the server hierarchy based on a  
3     number of connections of the existing server.

1           18.     The article of manufacture of claim 13, further comprising, upon receiving a request  
2     to delete an existing server in the hierarchy, deleting that server.

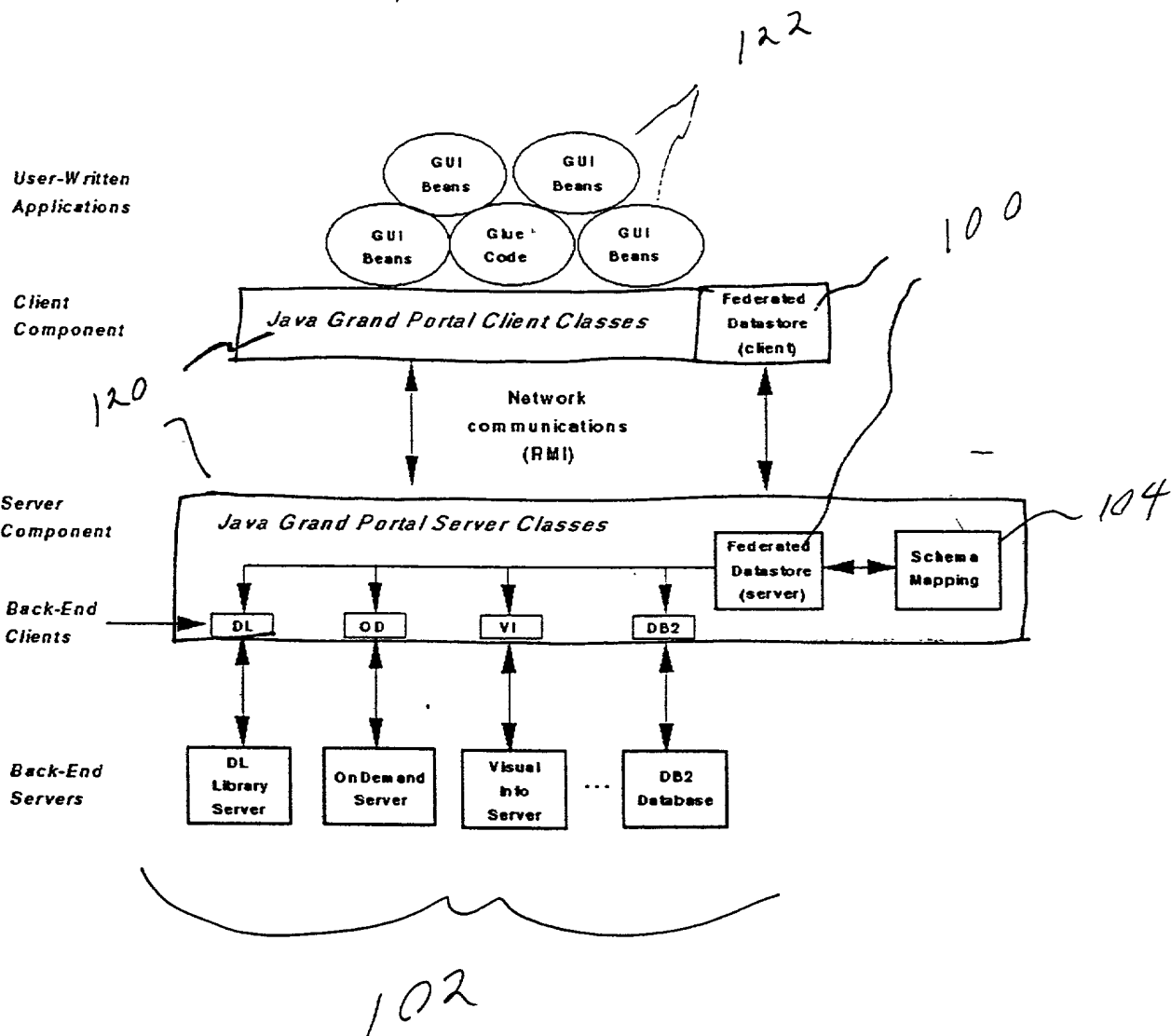
THE ARCHITECTURE AND IMPLEMENTATION OF A DYNAMIC RMI SERVER  
CONFIGURATION HIERARCHY TO SUPPORT FEDERATED SEARCH  
AND UPDATE ACROSS HETEROGENEOUS DATASTORES

ABSTRACT

An architecture and implementation of a dynamic RMI server configuration hierarchy to support federated search and update across heterogeneous datastores. The RMI server configuration hierarchy supports searching for data in one or more heterogeneous data sources within a computer system. A request for data is received at a federated data source. Then, a server is selected to process the request based on a load of the server and based on whether the server can satisfy the request for data.

667260"96966E60

FIG. 1



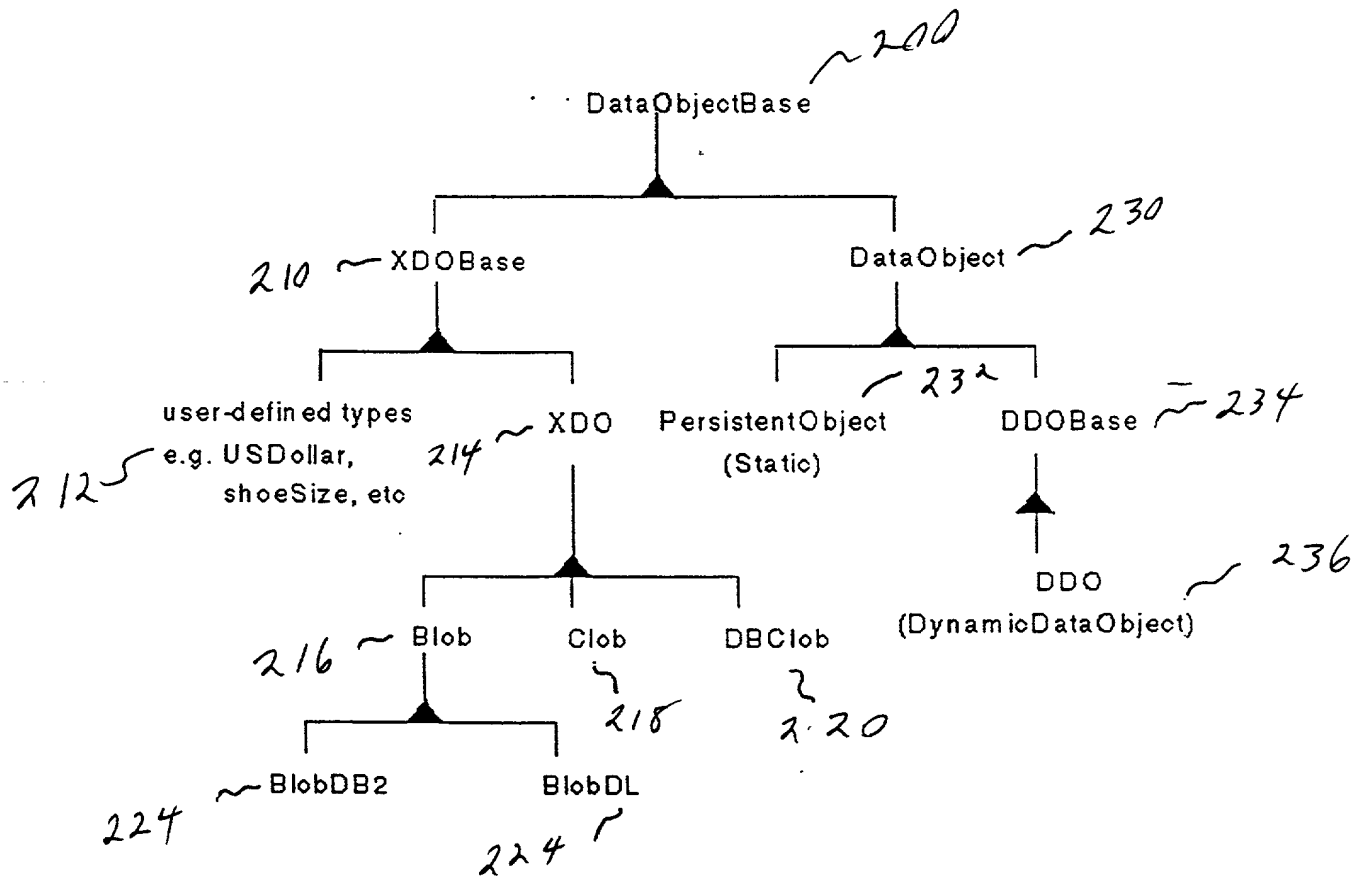
[illegible]

FIG. 3

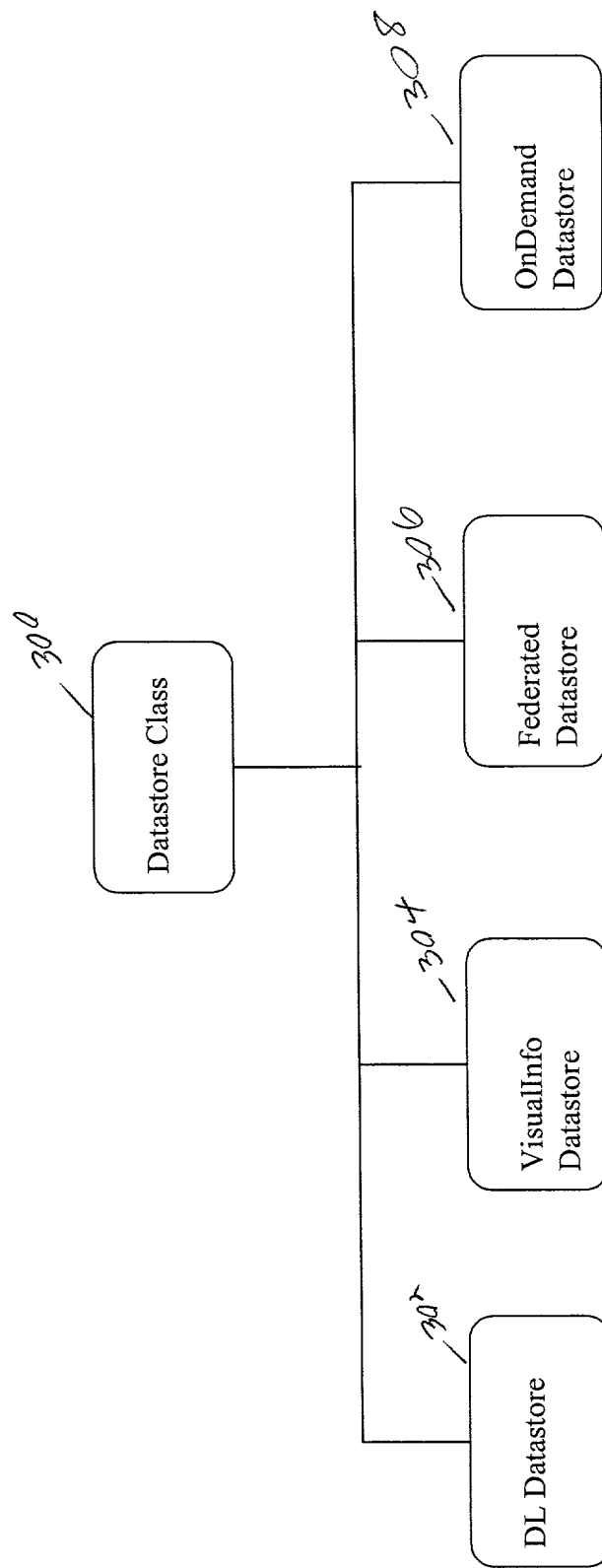


FIG. 4

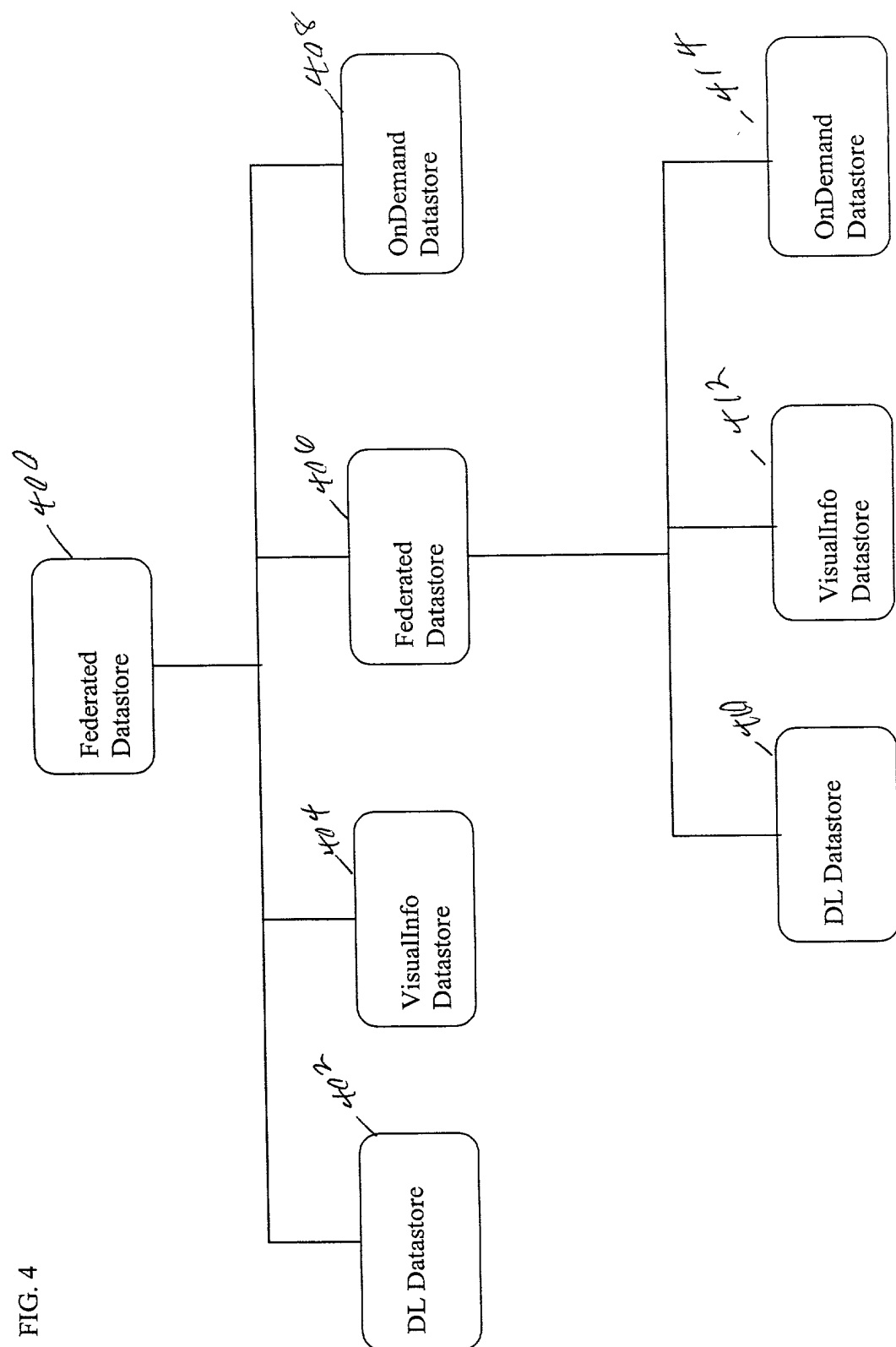




Fig. 5

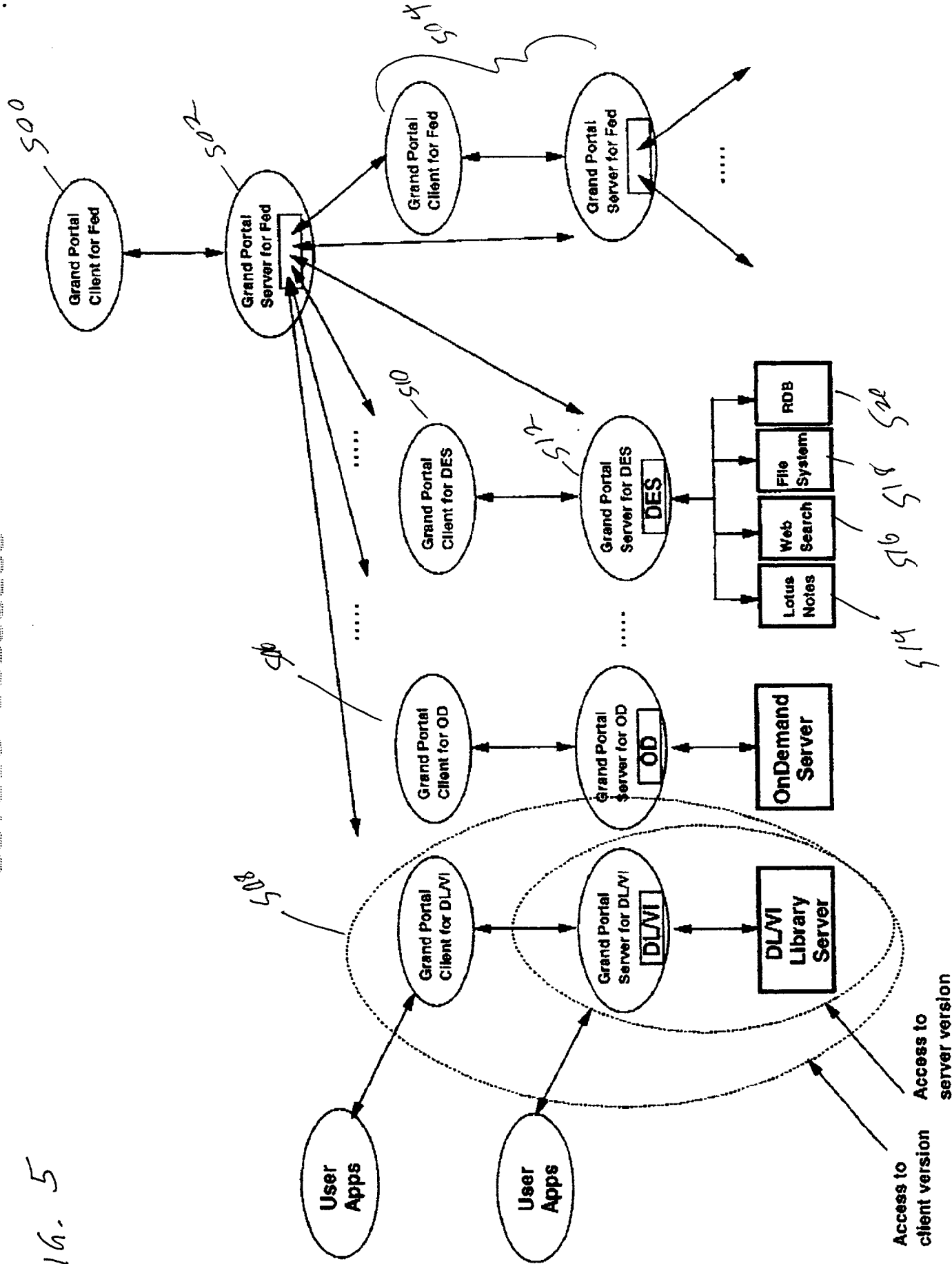


Fig. 6

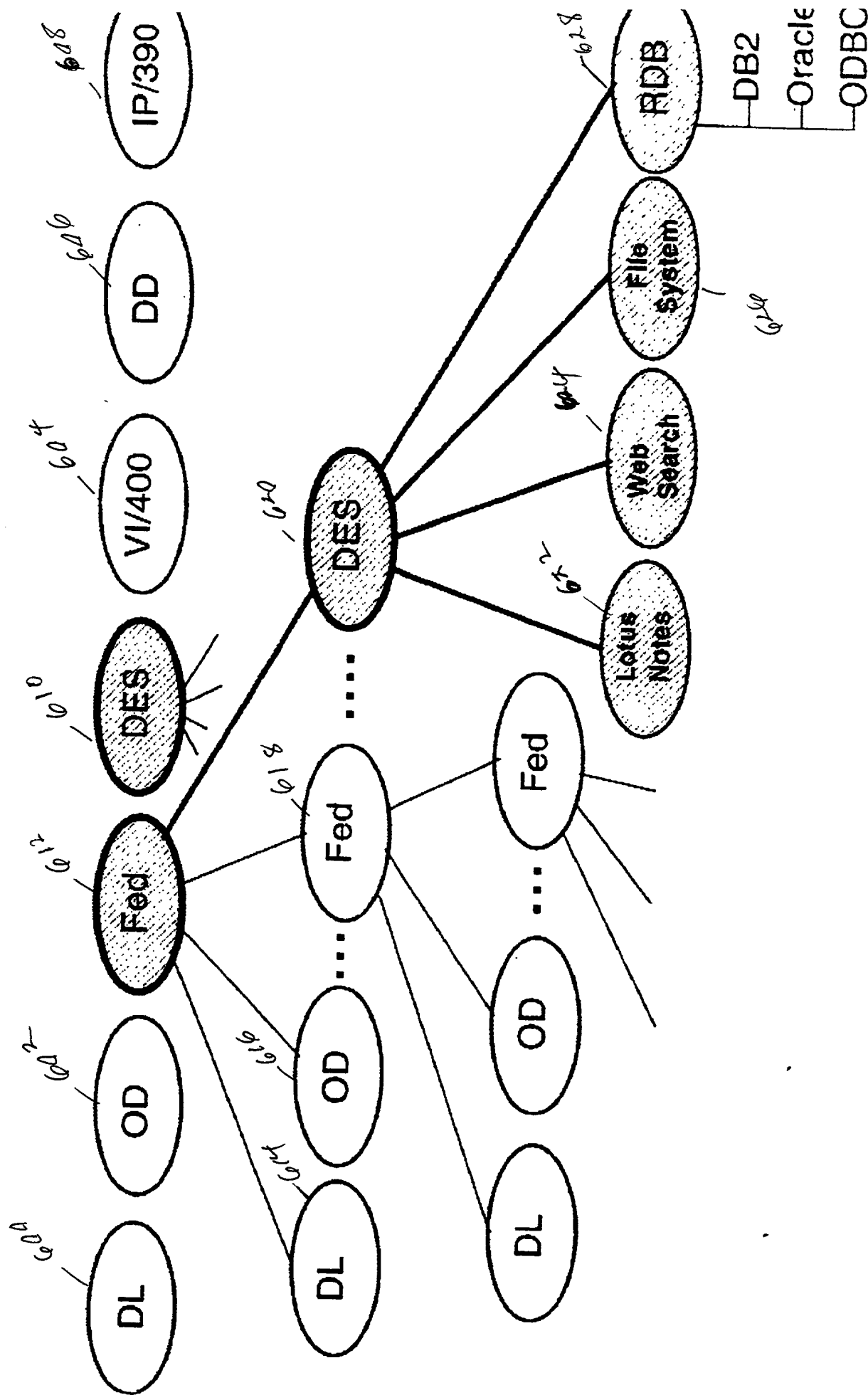




FIG. 8

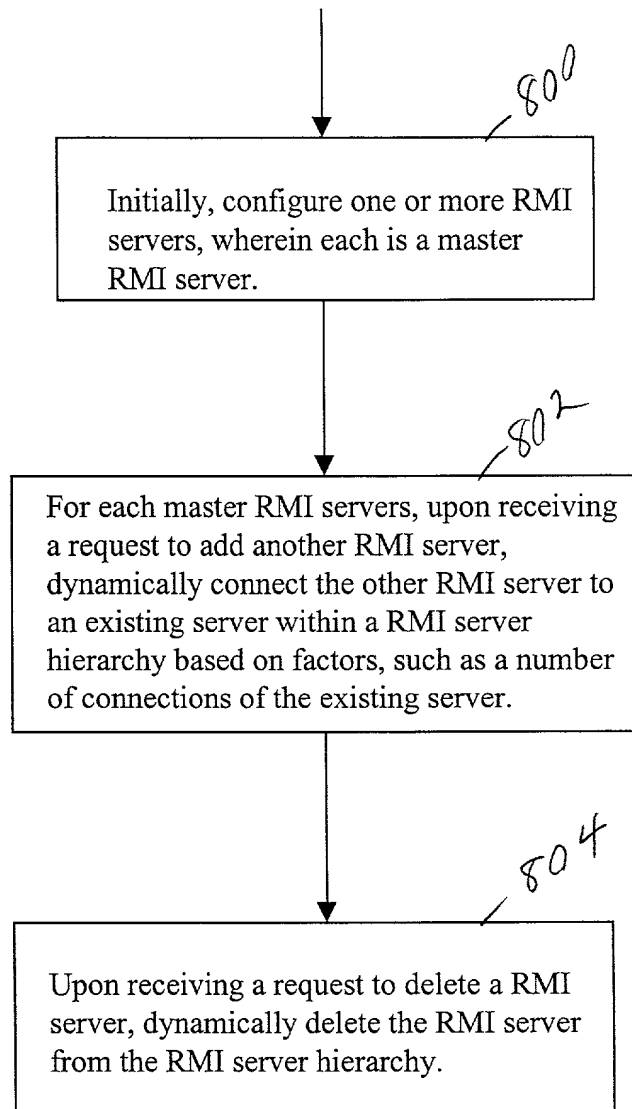
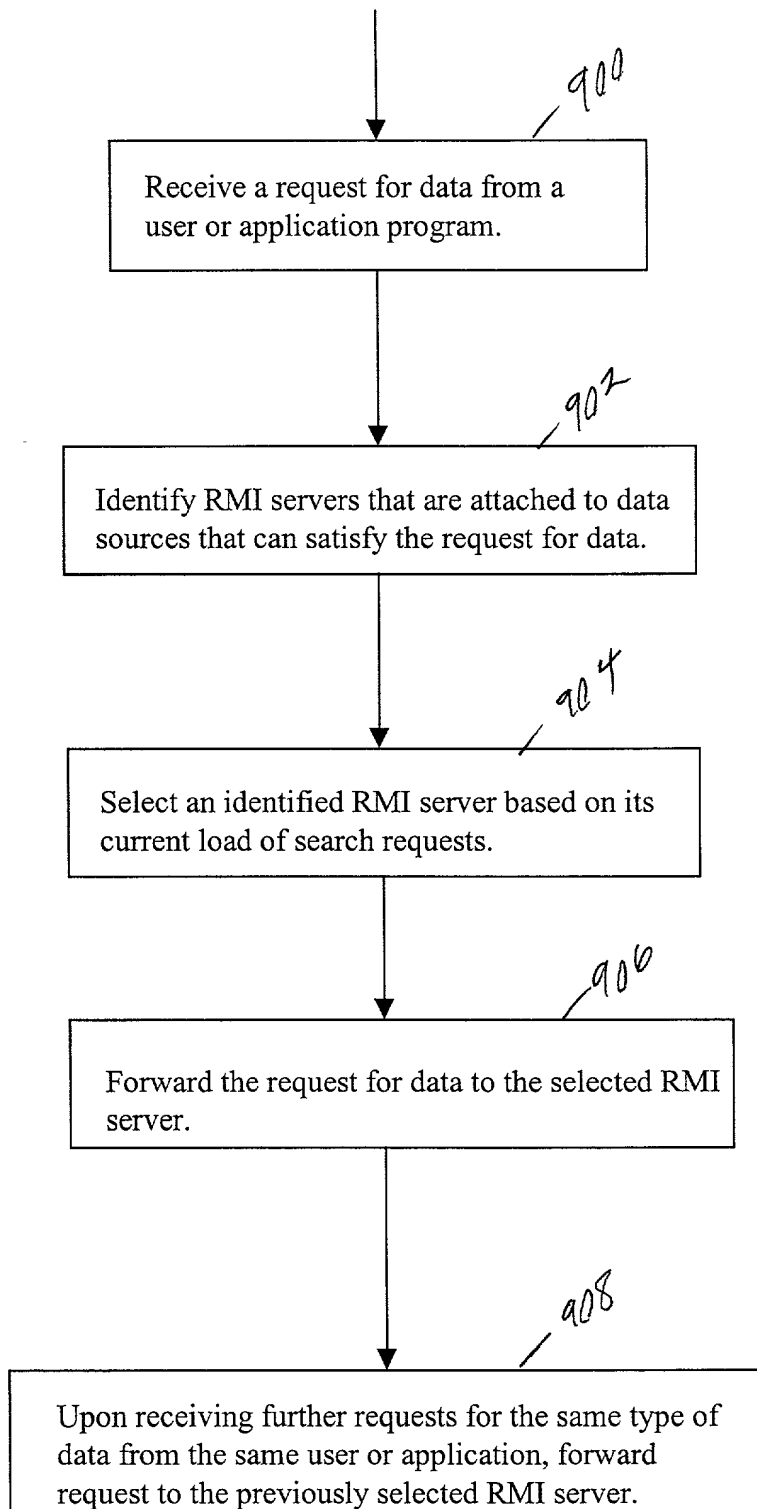


FIG. 9



## DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

DOCKET: ST999093

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

THE ARCHITECTURE AND IMPLEMENTATION OF A DYNAMIC RMI SERVER CONFIGURATION HIERARCHY TO SUPPORT FEDERATED SEARCH AND UPDATE ACROSS HETEROGENEOUS DATASTORES

the specification of which (check one)

X is attached hereto.  
 \_\_\_\_\_ was filed on \_\_\_\_\_  
 as Application Serial No. \_\_\_\_\_  
 and was amended on \_\_\_\_\_ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed
<u>None</u>	_____	_____	<u>Yes</u> <u>No</u>
(Number)	(Country)	(Day/Month/Year Filed)	

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56, which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

<u>None</u>	_____	_____
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

## DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

DOCKET: ST999093

**POWER OF ATTORNEY:** As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. (list name and registration number)

Romualdas Strimaitis, Registration No. 35,697, Prentiss W. Johnson, Registration No. 33,123, Ingrid M. Foerster, Registration No. 36,511, Timothy M. Farrell, Registration No. 37,321, Christopher A. Hughes, Registration No. 26,914, John E. Hoel, Registration No. 26,279, Edward A. Pennington, Registration No. 32,588, and Joseph C. Redmond, Jr., Registration No. 18,753, all of whom are attorneys with IBM Corporation; and

Edward G. Poplawski, Registration No. 33,439, Laurence H. Pretty, Registration No. 25,312, Robert A. Schroeder, Registration No. 25,373, Janaki Komanduri, Registration No. 40,684, Denise L. McKenzie, Registration No. 43,790, and Eric T. King, Registration No. 44,188, all of whom are attorneys with the law firm of Pretty, Schroeder & Poplawski.

Send correspondence to: Janaki Komanduri, Esq.  
Pretty, Schroeder & Poplawski  
444 S. Flower Street, 19th Floor  
Los Angeles, California 90071

Direct Telephone Calls to: (name and telephone number) Janaki Komanduri, (213) 622-7700

Full name of sole or first joint-inventor: Kehsing J. Chou

Inventor's signature:

Date:

Residence: 1075 Fleetwood Drive, San Jose, California 95120

Citizenship: U.S.A.

Post Office Address: Same

Full name of second joint-inventor: Mei-Ing W. Huang

Inventor's signature:

Date:

Residence: 7004 Quail Cliff Way, San Jose, California 95120

Citizenship: U.S.A.

Post Office Address: Same

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

DOCKET: ST999093

Full name of third joint-inventor: Taejae Lee

Inventor's signature:

Date:

Residence: 959 November Dr., Cupertino, California 95014

Citizenship: U.S.A.

Post Office Address: Same

Full name of fourth joint-inventor: Basuki N. Soetarman

Inventor's signature:

Date:

Residence: 309 DeSoto Drive, Los Gatos, California 95032

Citizenship: India

Post Office Address: Same

Full name of fifth joint-inventor: Robert Nelson Summers

Inventor's signature:

Date:

Residence: 5011 Carter Avenue, San Jose, California 95118

Citizenship: U.S.A.

Post Office Address: Same



DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

DOCKET: ST999093

---

Full name of sixth joint-inventor: Mimi Phuong-Thao Vo

---

Inventor's signature:

Date:

---

Residence: 6355 Hematite Ct., San Jose, California 95135

---

Citizenship: U.S.A.

---

Post Office Address: Same

---

09/26/99 09:26:50